Scheduling in Xen: Present and Near Future

Dario Faggioli
dario.faggioli@citrix.com

Cambridge – 27th of May, 2015
Introduction
Welcome

- Hello, my name is Dario
- I’m with Citrix since 2011 (in the Xen Platform Team)
Welcome (cont.)

Not often around here in 101... Because I work from home!
A bit of a Background (Check ;-) 

- Computer Engineering MSc → Ph.D on Real-Time Scheduling
- Scheduling already... but OS scheduling!
- What about virtualization:

[*] a well known benevolent dictator
A bit of a Background (Check ;-)

- Computer Engineering MSc → Ph.D on Real-Time Scheduling
- Scheduling already... but OS scheduling!
- What about virtualization:

  “I’m not a virtualization kind of guy. I think virtualization is evil”[*]

[*] a well known benevolent dictator
Me and Scheduling

- Not too good at nonsensical math

\[ \forall R_j \mid \tau_i \in \Gamma_j, I_i^j = \sum_{k | \tau_k \in \Phi_i^j} \xi_k(R_j) + \bigcup \Omega_i^j \quad (2) \]

and

\[ I_i = \sum_{j | \tau_i \in \Gamma_j} I_i^j \quad (3) \]

- Focused on implementing Real-Time scheduling algo-s in real-world OSes, such as Linux
- Tried to implement Earliest Deadline First (EDF) algorithm and have it merged upstream
- Attempted by ‘academicians’ a few times, just to blame the Linux community upon failure!
Me and Scheduling (cont.)

- Real-Time scheduling: "a solution to the wrong problem":
  - most of the time, there even is no overbooking
  - when there’s overbooking, not all activities are equally important
  - I/O is a bigger issue
Me and Scheduling (cont.)

- Real-Time scheduling: “a solution to the wrong problem”:
  - most of the time, there even is no overbooking
  - when there’s overbooking, not all activities are equally important
  - I/O is a bigger issue

- And in fact, one day in Boston, while presenting my work at the 2010 Kernel Summit...
Me and Scheduling (cont.)

- Real-Time scheduling: "a solution to the wrong problem":
  - most of the time, there even is no overbooking
  - when there’s overbooking, not all activities are equally important
  - I/O is a bigger issue

- And in fact, one day in Boston, while presenting my work at the 2010 Kernel Summit...

  "Real-Time is bull$*it![*]

  [*]the same well known benevolent dictator as before
Scheduling in Xen’s World
Scheduling and Xen

- There is **pretty much always** overbooking
- All activities (i.e., all VMs) are (or at least could be) **equally** important
There is **pretty much always** overbooking

All activities (i.e., all VMs) are (or at least could be) **equally important**
Scheduling and Xen

- There is **pretty much always** overbooking
- All activities (i.e., all VMs) are (or at least could be) **equally important**

... 

- I/O is **still** more important!
Wait a Second...

We are special!

- Xen is not a GPOS which can be turned into an hypervisor
- Xen’s scheduler needs to deal only with VMs’ vCPUs
Wait a Second...

We are special!

- Xen is not a GPOS which can be turned into an hypervisor
- Xen’s scheduler needs to deal only with VMs’ vCPUs

“I already told you, this isn’t ever going to happen. You do _NOT_ put a for_each_online_cpu() loop in the middle of schedule().

You also do not call stop_one_cpu(migration_cpu_stop) in schedule to force migrate the task you just scheduled to away from this cpu. That’s retarded.

Nacked-by: Peter Zijlstra <a.p.zijlstra@chello.nl>”
Wait a Second... (cont.)

Our I/O needs (CPU) scheduling!

- **Control Domain**: NetBSD or Linux
- **Driver Domain**
  - *device model (qemu)*
  - *toolstack*
  - Hardware Drivers
  - netback
  - blkback
- **Paravirtualized (PV) Domain**: NetBSD or Linux
  - *netfront*
  - *blkfront*
- **Xen Hypervisor**

- **Control Domain**: NetBSD or Linux
- **Driver Domain**
  - *device model (qemu)*
  - *toolstack*
  - Hardware Drivers
  - netback
  - blkback
- **Paravirtualized (PV) Domain**: NetBSD or Linux
  - *netfront*
  - *blkfront*
- **Xen Hypervisor**

- **Control Domain**: Fully Virtualized (HVM) Domain: Windows, FreeBSD...
- **Fully Virtualized (HVM) Domain**
Xen’s Scheduler Features
Hard and Soft Affinity

- pinning: you can run there and only there!
  #xl vcpu-pin vm1 0 2

- hard affinity: you can’t run outside of that spot
  # xl vcpu-pin vm1 all 8-12

- soft affinity: you can’t run outside of that spot and, preferably, you should run there
  # xl vcpu-pin vm1 all - 10,11

Same achieved with cpus= and cpus_soft= in config file.

cpus= or cpus_soft= in config file control where memory is allocated
### Hard and Soft Affinity (cont.)

```bash
root@tg03:~# xl vcpu-pin vml 0 2
root@tg03:~# xl vcpu-list vml
<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>VCPU</th>
<th>CPU State</th>
<th>Time(s)</th>
<th>Affinity (Hard / Soft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vml</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>1.5</td>
<td>2 / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>1</td>
<td>15</td>
<td>1.9</td>
<td>all / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>2</td>
<td>13</td>
<td>0.8</td>
<td>all / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>3</td>
<td>23</td>
<td>0.8</td>
<td>all / all</td>
</tr>
</tbody>
</table>

root@tg03:~# xl vcpu-pin vml all 8-12
root@tg03:~# xl vcpu-list vml
<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>VCPU</th>
<th>CPU State</th>
<th>Time(s)</th>
<th>Affinity (Hard / Soft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vml</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>1.5</td>
<td>8-12 / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>1.9</td>
<td>8-12 / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>2</td>
<td>9</td>
<td>0.8</td>
<td>8-12 / all</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>0.8</td>
<td>8-12 / all</td>
</tr>
</tbody>
</table>

root@tg03:~# xl vcpu-pin vml all -10,11
<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>VCPU</th>
<th>CPU State</th>
<th>Time(s)</th>
<th>Affinity (Hard / Soft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vml</td>
<td>5</td>
<td>0</td>
<td>11</td>
<td>1.5</td>
<td>8-12 / 10-11</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>1</td>
<td>10</td>
<td>1.9</td>
<td>8-12 / 10-11</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>2</td>
<td>11</td>
<td>0.8</td>
<td>8-12 / 10-11</td>
</tr>
<tr>
<td>vml</td>
<td>5</td>
<td>3</td>
<td>10</td>
<td>0.8</td>
<td>8-12 / 10-11</td>
</tr>
</tbody>
</table>
```
for Dom0

- `dom0_max_vcpu`: makes sense
- `dom0_vcpus_pin`: bleah!!
- `dom0_nodes`: new parameter. Place Dom0’s vCPUs and memory on one or more nodes
  - strict (default) uses hard affinity
  - relaxed uses soft affinity
More about NUMA

Automatic placement policy in libxl (since Xen 4.2)

- acts at domain creation time
- easy to tweak (at libxl build time, for now) heuristics:
  - use the smallest possible set of nodes (ideally, just one)
  - use the (set of) node(s) with fewer vCPUs bound to it ([will] consider both hard and soft affinity)
  - use the (set of) node(s) with the most free RAM (mimics the “worst fit” algorithm)
More about NUMA

Automatic placement policy in libxl (since Xen 4.2)

- acts at domain creation time
- easy to tweak (at libxl build time, for now) heuristics:
  - use the smallest possible set of nodes (ideally, just one)
  - use the (set of) node(s) with fewer vCPUs bound to it ([will] consider both hard and soft affinity)
  - use the (set of) node(s) with the most free RAM (mimics the “worst fit” algorithm)

Example:
4 GB VM, 6 GB free on node0, 8 GB free on node1
More about NUMA

Automatic placement policy in libxl (since Xen 4.2)

- acts at domain creation time
- easy to tweak (at libxl build time, for now) heuristics:
  - use the smallest possible set of nodes (ideally, just one)
  - use the (set of) node(s) with fewer vCPUs bound to it ([will] consider both hard and soft affinity)
  - use the (set of) node(s) with the most free RAM (mimics the “worst fit” algorithm)

Example:
4 GB VM, 6 GB free on node0, 8 GB free on node1
Bound vCPUs: 12 to node0, 16 to node1 \(\Rightarrow\) use node0
More about NUMA

Automatic placement policy in libxl (since Xen 4.2)

- acts at domain creation time
- easy to tweak (at libxl build time, for now) heuristics:
  - use the smallest possible set of nodes (ideally, just one)
  - use the (set of) node(s) with fewer vCPUs bound to it ([will] consider both hard and soft affinity)
  - use the (set of) node(s) with the most free RAM (mimics the “worst fit” algorithm)

Example:
4 GB VM, 6 GB free on node0, 8 GB free on node1
Bound vCPUs: 12 to node0, 16 to node1 ⇒ use node0
Bound vCPUs: 8 to node0, 8 to node1 ⇒ use node1
More about NUMA

Automatic placement policy in libxl (since Xen 4.2)

- acts at domain creation time
- easy to tweak (at libxl build time, for now) heuristics:
  - use the smallest possible set of nodes (ideally, just one)
  - use the (set of) node(s) with fewer vCPUs bound to it ([will] consider both hard and soft affinity)
  - use the (set of) node(s) with the most free RAM (mimics the “worst fit” algorithm)

Example:
4 GB VM, 6 GB free on node0, 8 GB free on node1
Bound vCPUs: 12 to node0, 16 to node1 \(\rightarrow\) use node0
Bound vCPUs: 8 to node0, 8 to node1 \(\rightarrow\) use node1

Coming: node distances, IONUMA, vNUMA
Workin’ On
Exploiting Intel PSR

Intel Platform Shared Resource Monitoring (PSR):
- Cache Monitoring Technology (CMT)
- Memory Bandwidth Monitoring (MBM)

Tells how much cache/mem. bandwidth is being consumed by a certain 'activity' running on a CPU. E.g., about CMT:
Exploiting Intel PSR

Intel Platform Shared Resource Monitoring (PSR):
- Cache Monitoring Technology (CMT)
- Memory Bandwidth Monitoring (MBM)

Tells how much cache/mem. bandwidth is being consumed by a certain 'activity' running on a CPU. E.g., about CMT:

Cool, eh? Oh, well:
- moderately accurate and fast:
  hey, it’s done in hardware after all! :-)
- limited in scope and not very flexible:
  heh, it’s done in hardware after all! :-(

Cambridge – 27th of May, 2015
Exploiting Intel PSR (cont.)

Current CMT support:

- in Linux (and hence KVM): cache usage stats for tasks and group of tasks
Exploiting Intel PSR (cont.)

Current CMT support:
- in Linux (and hence KVM): cache usage stats for tasks and group of tasks

Can it be used in more clever ways, e.g., in the scheduler?
Exploiting Intel PSR (cont.)

Current CMT support:

- in Linux (and hence KVM): cache usage stats for tasks and group of tasks

Can it be used in more clever ways, e.g., in the scheduler?

- in Linux: Yes... as soon as hell freezes!
Exploiting Intel PSR (cont.)

Current CMT support:

▶ in Linux (and hence KVM): cache usage stats for tasks and group of tasks

Can it be used in more clever ways, e.g., in the scheduler?

▶ in Linux: Yes... as soon as hell freezes!
▶ in Xen: Yes! (Or, at least, nothing stops us trying)
Exploiting Intel PSR (cont. II)

Biggest limitations:

- num. of activities that can be monitored is limited
- applies to L3 cache only, for now
Exploiting Intel PSR (cont. II)

Biggest limitations:

- num. of activities that can be monitored is limited
- applies to L3 cache only, for now

What would be desirable:

- per-vCPU granularity $\Rightarrow$ No! Too few monitoring IDs
- L2 occupancy/bandwidth stats, for helping intra-socket scheduling decisions $\Rightarrow$ No! Only L3
Exploiting Intel PSR (cont. II)

Biggest limitations:
- num. of activities that can be monitored is limited
- applies to L3 cache only, for now

What would be desirable:
- per-vCPU granularity → No! Too few monitoring IDs
- L2 occupancy/bandwidth stats, for helping intra-socket scheduling decisions → No! Only L3

What I’m thinking to:
- use one monitoring ID per pCPU. This gives:
  - how ‘cache hungry’ a pCPU is being
  - how much free cache there is on each socket/NUMA node
- sample periodically and use for mid-level load balancing decisions
- ... ideas welcome!!
Credit2 scheduler, authored by George, is still in experimental status.
Credit2 scheduler, authored by George, is still in experimental status.

Take it out from there!!
Credit2 scheduler, authored by George, is still in experimental status.

**Take it out from there!!**

What’s missing:

- SMT awareness (done, missing final touches)
- hard and soft affinity support (someone working on it)
- tweaks and optimization in the load balancer (someone looking at it)
- cap and reservation (!!!)
Credit2 scheduler, authored by George, is still in experimental status.

Take it out from there!!

What’s missing:

- SMT awareness (done, missing final touches)
- hard and soft affinity support (someone working on it)
- tweaks and optimization in the load balancer (someone looking at it)
- cap and reservation (!!!)

Plan: mark it as !experimantal for 4.6, make it default for 4.7 (let’s see...)
Credit2: Why?

Schedulers do age: as they grow old, they tend to grow “hacks”
Credit2: Why?

Schedulers *do age*: as they grow old, they tend to grow “hacks”

- Seen with the Linux scheduler:
Credit2: Why?

Schedulers *do age*: as they grow old, they tend to grow “hacks”

- Seen with the Linux scheduler:
  - Once upon a time, there was the O(1) scheduler, then...
Credit2: Why?

Schedulers *do age*: as they grow old, they tend to grow “hacks”

- Seen with the Linux scheduler:
  - Once upon a time, there was the O(1) scheduler, then...
  - Once upon a time (again!), there was CFS, then...
Credit2: Why?

Schedulers do age: as they grow old, they tend to grow “hacks”

- Seen with the Linux scheduler:
  - Once upon a time, there was the O(1) scheduler, then...
  - Once upon a time (again!), there was CFS, then...

- Less true with Credit... still:
Credit2: Why?

Schedulers *do age*: as they grow old, they tend to grow “hacks”

- Seen with the Linux scheduler:
  - Once upon a time, there was the O(1) scheduler, then...
  - Once upon a time (again!), there was CFS, then...

- Less true with Credit... still:
  - CSCHED_PRI_TS_BOOST sort of falls into this
Credit2: Why?

Schedulers do age: as they grow old, they tend to grow “hacks”

▶ Seen with the Linux scheduler:
  ▶ Once upon a time, there was the O(1) scheduler, then...
  ▶ Once upon a time (again!), there was CFS, then...

▶ Less true with Credit... still:
  ▶ CSCHED_PRI_TS_BOOST sort of falls into this
  ▶ any addition, at this stage, would fall into this (e.g., load balancing based on historical load)
Credit2: Why? (cont.)

Complexity:
Credit2: Why? (cont.)

Complexity:

▶ in Credit we have:
  ▶ credits and weights
  ▶ 2 priorities
  ▶ oh, actually, it’s 3
  ▶ active and non active state of vCPUs
  ▶ flipping between active/non-active means flipping between burning/non-burning credits, which in turns means wandering around among priorities
Credit2: Why? (cont.)

Complexity:

- in Credit we have:
  - credits and weights
  - 2 priorities
  - oh, actually, it’s 3
  - active and non active state of vCPUs
  - flipping between active/non-active means flipping between burning/non-burning credits, which in turns means wandering around among priorities

- in Credit2 we have:
  - credits burned basing on weights
Credit2: Why? (cont. II)

Complexity (II):
- in Credit we have:
  - credits-per-msec, timeslice, ticks-per-timeslice
  - can we change the timeslice? Yes, of course... **in theory!**
Credit2: Why? (cont. II)

Complexity (II):

- in Credit we have:
  - credits-per-msec, timeslice, ticks-per-timeslice
  - can we change the timeslice? Yes, of course... in theory!

- in Credit2 we have:
  - no timeslice at all (just min-timer, max-timer)
Credit2: Why? (cont. II)

Complexity (an example): start_time
Credit2: Why? (cont. II)

Complexity (an example): start_time

in Credit we have:

```c
s_time_t start_time; /* When we were scheduled (used for credit) */
svc->start_time += (credits * MILLISECS(1)) / CSCHED_CREDITS_PER_MSEC;
scurr->start_time -= now;
snext->start_time += now;
snext->start_time += now;
```
Credit2: Why? (cont. II)

Complexity (an example): start_time

in Credit we have:

```c
s_time_t start_time; /* When we were scheduled (used for credit) */
svc->start_time += (credits * MILLISECS(1)) / CSCHED_CREDITS_PER_MSEC;
scurr->start_time -= now;
snext->start_time += now;
snext->start_time += now;
```

in Credit2 we have:

```c
s_time_t start_time; /* When we were scheduled (used for credit) */
svc->start_time = now;
delta = now - svc->start_time;
svc->start_time = now;
snext->start_time = now;
```
Credit2: Why? (cont. III)

Scalability:
Credit2: Why? (cont. III)

Scalability:

- in Credit
  - periodic runqueue sorting. *Freezes* a runqueue
  - periodic accounting. *Freezes* the whole scheduler!
Credit2: Why? (cont. III)

Scalability:

- in Credit
  - periodic runqueue sorting. *Freezes* a runqueue
  - periodic accounting. *Freezes* the whole scheduler!

- in Credit2 we have:
  - “global” lock only for load balancing
    (looking at improving it)
In general, more advanced, a lot of potential:

- historical load based load balancing
- runqueue kept in order of credit (instead than Round-Robin as in Credit1)
- configurable runqueue arrangement
Credit2: Why (cont. V)

Performance?
Performance? Some tweaks still missing, but really promising.
Performance? Some tweaks still missing, but really promising.
Driver Domain Aware Scheduling

Suppose:

- vCPU $x$ is top priority (higher credits, whatever)
- vCPU $x$ issues an I/O operation. It has some remaining timeslice (or credit, or whatever) available, but it blocks waiting for results
- some other domains' vCPUs $y$, $w$ and $z$ have higher priority than I/O's vCPUs (Dom0 or driver domain)
Driver Domain Aware Scheduling

Suppose:

- vCPU $x$ is top priority (higher credits, whatever)
- vCPU $x$ issues an I/O operation. It has some remaining timeslice (or credit, or whatever) available, but it blocks waiting for results
- some other domains’ vCPUs $y$, $w$ and $z$ have higher priority than I/O’s vCPUs (Dom0 or driver domain)

Schedule: $v_x$, $v_y$, $v_w$, $v_z$, $v_{drv\_dom}$ $\rightarrow$ only now $v_x$ can resume
Driver Domain Aware Scheduling (cont.)

What if, \( v_x \) could donate its timeslice to the entity that is blocking it?
Driver Domain Aware Scheduling (cont.)

What if, \( v_x \) could donate its timeslice to the entity that is blocking it?

Schedule: \( v_x, v_{\text{drv\_dom}}, v_x, v_w, v_z \rightarrow v_x \) unblocks right away (this, assuming servicing I/O to be quick, and does not even exhaust \( v_x \) timeslice)
Driver Domain Aware Scheduling (cont.)

What if, $v_x$ could *donate* its timeslice to the entity that is blocking it?

Schedule: $v_x$, $v_{drv\_dom}$, $v_x$, $v_w$, $v_z \rightarrow v_x$ unblocks right away (this, assuming servicing I/O to be quick, and does not even exhaust $v_x$ timeslice)

- avoids priority inversion (no, we’re not the Mars Pathfinder, but still...)
- makes $v_x$ sort of “pay”, from the CPU load it generates with its I/O requests (fairness++)
Conclusions
Conclusions

Scheduling: we **probably** are doing fine...
Scheduling: we *probably* are doing fine... Maybe at least *not too bad*?
Conclusions

Scheduling: we *probably* are doing fine... Maybe at least *not too bad*?

However:
  ➤ we should assess whether that is the case or not (for as many workloads as we possibly can)
Scheduling: we *probably* are doing fine... Maybe at least *not too bad*?

However:

- we should assess whether that is the case or not (for as many workloads as we possibly can)
- even if yes,
Conclusions

Scheduling: we **probably** are doing fine... Maybe at least *not too bad*?

However:

- we should assess whether that is the case or not (for as many workloads as we possibly can)
- even if yes, we should do even better!
Thanks again,

Questions?